FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

"Out of band" authentication using an Android device

Luiz Fernando Vilela Cruz

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: José Manuel de Magalhães Cruz

June 27, 2016

Resumo

Com a expansão de serviços na "nuvem" e com o aumento na variedade dos serviços disponíveis na Internet, as pessoas nunca tiveram uma presença digital tão grande. Com isto agora é ainda mais importante que os usuários destes serviços se preocupem com os métodos ao qual acessam suas contas, se certificando que elas estejam sempre seguras.

O método preferencial para se autenticar atualmente envolve a digitação de senhas, que as vezes são complementadas com diferentes fatores de autenticação para aumentar a segurança no caso de serem interceptadas. Mas é comum que os usuários usem as mesmas senhas para acessar diferentes contas, aumentando os riscos que credenciais roubadas possam resultar no acesso indevido de muitas contas (que não usam um segundo fator de autenticação). A necessidade de se fornecer um nome de usuário durante o processo também cria o risco de que isto seja utilizado para monitorar as ações dos usuários, quebrando sua privacidade.

Outro problema atual envolve a autenticação em um dispositivo ao qual não se pode confiar que seja seguro, como um computador público. Nestes casos, sempre que quiser acessar uma de suas contas você é forçado a fornecer suas credenciais, e pode apenas ter a esperança de que elas não sejam roubadas no processo.

Uma das dificuldades em se resolver estes problemas vem do fato de que existem um grande número de diferentes soluções que foram propostas para melhorar a segurança do processo, mas que não são compatíveis entre si. Isto pode forçar os usuários a escolher entre adotar múltiplos métodos de autenticação para acessar todos os seus serviços, ou de desistir e voltar a ter que memorizar e digitar senhas.

Uma das atuais ideias propostas para resolver estes problemas já resultou em várias produtos comerciais, utilizando um aplicativo em um *smartphone* para armazenar os segredos do usuário (normalmente uma chave privada RSA) e para autentica-lo. Estes aplicativos utilizam a câmera do dispositivo para ler um código QR, de modo a obter um desafio numérico que é digitalmente assinado e enviado através da internet. Utilizar um canal paralelo traz a vantagem de não se ter de confiar mais no dispositivo ao qual se está autenticando, que pode não ser seguro. É importante que os usuários tenham o controle sobre suas próprias identidades digitais, confiando o mínimo possível nos serviços e dispositivos que utilizam para se autenticar.

Nesta dissertação estas soluções que utilizam *smartphones* são brevemente analisadas com o objetivo de se ter uma base de comparação com a solução proposta neste trabalho, que também utiliza um *smartphone* para ler um código QR, mas que tem o foco principal de dar aos usuários o máximo de controle sobre o processo de autenticação. Este objetivo foi obtido com o desenvolvimento de um simples protocolo de comunicação baseado em texto, de maneira que fosse simples de ser analisado, e também com a utilização de criptografia de chave pública RSA e evitando utilizar autoridades de certificação, procurando ter que confiar o mínimo possível em terceiros. Um servidor, página web e aplicativo Android foram desenvolvidos como uma prova de conceito, com todo o código e protocolos desenvolvidos sendo disponibilizados na Internet com licenças abertas.

Abstract

With the expansion of "cloud" services and the increased offer of a variety of services online, people now have a much larger digital presence than before. This means that it is now even more important for internet users to be concerned with how they access their accounts, making sure that they are both secure and private.

The currently preferred method of authentication normally involves typing passwords, or sometimes combining them with a different authentication factor to increase security in case they are stolen. But users often use the same passwords for multiple services, which increases the risk that a stolen password could result in multiple accounts (that don't use a second factor) becoming compromised. The fact that users are required to type in their username during the process also puts them at risk of having their actions tracked against their will.

Another issue comes from authenticating on a device that you do not own or trust, such as a public computer terminal. In these cases, whenever you need to access one of your accounts you are forced to input your credentials and hope that they are not being stolen in the process.

One of the difficulties in solving these problems comes from the fact that there are a large number of solutions that promise better security, but that are usually not interoperable or widely adopted. This can force users to either use multiple methods to be able to safely authenticate themselves to all of their accounts, or to give up and go back to memorizing and typing passwords.

One of the current ideas to solve these issues has resulted in a number of commercial implementations, usually involving the use of a smartphone application to store the user's secrets (usually an RSA private key) and to authenticate the user. These applications can use the device's camera to read a QR code in order to obtain a challenge, which is then digitally signed and sent through the internet. Using a parallel or "out of band" channel has the advantage of not relying on the connection from the device you are authenticating to, which cannot be trusted to be secure. It is important to give users the control over their own digital identities, requiring them to put less trust on the services and devices they authenticate to.

In this dissertation these smartphone solutions are briefly analyzed in order to establish a base of comparison for my own solution, which was also developed to use a smartphone to read a QR code, but has the main focus of trying to give the users full control over the authentication process. This was achieved by developing a very simple text based communication protocol that could be easily analyzed, as well as by using RSA public key cryptography and avoiding certificate authorities in order to rely as little as possible on third parties. A server, web page and Android application were developed as a proof of concept, with all the code and protocols being released in an open source license.

Acknowledgements

I would like to thank my family for supporting me during the realization of this work, giving me the tools and the incentive required to get to the finish line. And also many thanks to my supervisor, Professor José Magalhães Cruz, for giving me the right feedback whenever it was necessary, and for guiding me to the right path.

Luiz Fernando Vilela Cruz

"If I have seen further,
it is by standing on the shoulders of giants."

Isaac Newton

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
app	Mobile application
AES	Advanced Encryption Standard
DNS	Domain Name Server
eSE	Embedded Secure Element
FIDO	Fast Identity Online
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HCE	Host Card Emulation
IP	Internet Protocol
NFC	Near Field Communication
OTP	One time password
PIN	Personal Identification Number
QR	Quick Response
SIM	Subscriber Identification Module
SDK	Software Development Kit
SQL	Structured Query Language
U2F	Universal 2nd Factor
UAF	Universal Authentication Framework
USB	Universal Serial Bus
URL	Uniform Resource Locator

Chapter 1

Introduction

With the gradual reduction in the costs of electronics and with the expanding popularity of the Internet, the number of available services that require some form of interaction with a digital device is rapidly increasing. In order to allow for more features and to facilitate control, service providers usually require their users to register themselves, asking them to provide personal information in the process. This has resulted in users commonly having a large number of digital accounts, that are often accessed through several different devices every day.

The most common method to authenticate yourself in order to access one of your accounts involves providing "something you know", which commonly takes the form of a username and password pair that has to be typed on the login form. The username is usually public and is used to identify the user, while the password is private and has the purpose of proving that the user is who he claims to be. The two other possible authentication methods (commonly called "factors of authentication") involve providing "something you have", such as a unique card or token, or providing "something you are", such as a fingerprint or retina scan. These other factors are much less common, mostly because they normally require additional hardware, increasing the complexity and cost of the process.

With the popularization of "cloud" services through the Internet, it is also ever more common for users to be able to access all their data and software from devices that they do not own or control. Using "cloud" data storage solutions has greatly increased the mobility and convenience for users, which no longer have to carry their own devices everywhere they go. Simple examples of this includes workers who are able to do their jobs from any computer-like device supplied by their employers, or students, who could just as easily access and read their e-mails or research materials from any computer in their universities. But this also means that users are now as vulnerable as ever to a large number of possible attacks, designed to gain access to their accounts and information.

1.1 Motivation

With users having an ever increasing number of accounts and credentials, and with the increase in the usefulness of any device with access to the "cloud", all problems related with account security and privacy have become even more damaging. Every time we type in our credentials in a device, we are forced to accept the fact that there is a risk of them being stolen. This theft can either happen in the device itself (by keyloggers or malware) or during it's transmission through the Internet. These risks are already considerable on devices we control and can care for, but are even bigger when we authenticate ourselves to devices such as work computers or cash machines, which we have no choice but to trust that are secure.

Since services are requiring ever more complex and lengthy passwords, users nowadays often have to rely on using the same credentials on multiple accounts in order to be able to remember them. This means that if a single one of those passwords is compromised, an attacker could gain access to multiple accounts and create even bigger problems for the user. Password managers can be used to facilitate memorizing your credentials, but they create a single point of failure and are generally difficult to use on devices you do not own.

For these reasons, the use of "something you know" as the only factor of authentication can no longer be considered safe. Many services already allow the use of a second factor, usually requiring you to confirm your login attempt by requesting you to type a one time password sent to your phone or e-mail, or by requiring you to use another device such as a smartcard or token. This could prevent attackers from gaining access to your accounts, but It does not prevent your credentials from being stolen, which often happens without your knowledge. This can be a problem if they are re-used to gain access to another of your accounts that shares the same password, or if this information is used to track you and your actions. Using multiple factors of authentication also has the disadvantages of increasing the complexity and cost of the authentication systems, and of limiting how fast users can authenticate themselves, which is why many users don't use them even if that option is available.

A number of different solutions have been proposed to replace passwords as the main form of authentication, with the objective of increasing our security and privacy, but most of them are either too insecure, too complicated or too expensive to be widely adopted. In order for a solution to become popular, it would have to be cheap, secure and simple enough to be used by almost anyone. Most proposed alternatives so far also encounter the problem of being proprietary technologies or protocols, which only makes them more expensive and harder to use.

Smartcards or tokens have been in use for many years, but are not very popular for general authentication due to their extra cost and the hassle of having to carry them wherever you go. Since they can require special equipment to be accessed, they are mostly used inside institutions or corporations that emit their own cards, which can rarely be used for other purposes. Biometrics also require special equipment on the device you want to authenticate yourself to, but are plagued by a number of privacy and security problems[4, 5, 6], which prevents them from being a full solution by themselves.

On the other hand, during the last decade new methods have been proposed that leverage the fact that most users already have a smartphone, which could be used in order to reduce costs. By using a mobile application to authenticate themselves, users could type their passwords on their devices instead, or could even avoid the use of passwords altogether by using challenge/response protocols. These solutions are specially useful when authenticating to potentially compromised computers, which could steal your password if it was typed on them.

These apps can greatly increase security because in principle a smartphone is less likely to be compromised than the device you want to authenticate to, of which you have no control over. Using your smartphone for authentication will become even more secure with the widespread adoption of embedded secure elements (eSE), which are predicted to be in 62% of the smartphones shipped by 2017[7]. This technology, along with host card emulation (HCE), could essentially transform every smartphone into a smartcard, helping to securely store information such as encryption keys and passwords.

Unfortunately none of these methods are flawless and have failed to obtain a large popularity, partially due to the lack of standardization and interoperability, which fragments the user base and makes each solution less useful. Some organizations, such as the FIDO alliance[8], have released specifications and standards in the hopes of improving the interoperability of these solutions, and contributing to the eventual replacement of using passwords as the main authentication method, but there is also no guarantee that these protocols will become the norm.

Finally, one thing that almost none of the available solutions do is being completely transparent and simple to understand by the end user, who ends up having to simply shift their trust from the devices they currently type their passwords into, to the companies that created these applications and protocols. Trusting a third party has some advantages, but with the constant news of data breaches on Internet company servers and of the abuse of our personal data by their practices and employees, it seems only logical to try to shift the user's responsibility for their own digital identities back to themselves. It is important to give people the tools to be able to consciously decide what they want to use to authenticate themselves. In order to allow that to happen, these solutions have to make sure that only the users have the capability to authenticate themselves to their accounts, while also allowing full transparency on how their applications and protocols work.

1.2 Objectives

The first objective of this work was to briefly analyze the available methods and protocols which promise to both hinder the collection of personal information during authentication, and to improve the security of user's accounts. This analysis was focused mostly on the solutions using smartphone applications with challenge/response protocols, in order to facilitate the second objective of this thesis.

The second objective was to select the best practices, methods and protocols encountered in order to develop and test the solution here proposed. This solution would have to use as much of the current infrastructure used by password authentication as possible, and also be simple enough

to facilitate its adoption and integration into existing services. In order to avoid introducing new devices to the authentication process, the choice of using smartphones to authenticate the user was made early on in the project.

The third and final objective was to release the code and protocols developed during this work with an open source license, in order to allow any institution or individual to easily implement them as they see fit. By facilitating the increase in the number of services that allow alternatives to password authentication, users would eventually come to expect them in all their accounts, compelling all service providers into adopting better solutions as well.

1.3 Document Structure

There are three other chapters besides the introduction. In Chapter 2 the state of the art related to user authentication is presented, analyzing the best found solutions that utilize smartcards, tokens, biometrics, smartphones or a mix of those. A comparison between them is also present in that chapter, taking into account their usability and how well they fare against a number of attacks.

Chapter 3 presents my solution to the problems mentioned during the introduction, starting with an explanation of the project requirements, followed by details on how all of it's components work.

Chapter 4 details the proof of concept implementation, detailing the developed communication protocol and briefly explaining how the demonstration web page, application and server are structured. It also presents an analysis of the results, with regards to it's security and usability, and comparing it to the solutions presented on chapter 2.

Chapter 5 presents the conclusions taken from this work, discussing what contributions were made by this dissertation and listing all the possible improvements that could be applied to the developed solution.

Chapter 2

State of the art

This chapter introduces the existing solutions that could be used to allow a user to authenticate without using passwords. The focus of this chapter will be on smartphone applications and solutions, but some research on smartcards will also be presented. It is divided into three sections, with the first one detailing the solutions that use smartcards or tokens, while the second section details the most promising smartphone applications and protocols available, providing a quick summary of each one's advantages and possible problems, both in relation to usability and security. The final section shows a comparison between the smartphone solutions.

Proposals that only use biometrics were not taken into consideration in part because they are not the focus of this work, but also because they are not secure enough on their own. Biometrics cannot be revoked or shared, can be easily stolen[9], can result in false positives or negatives and can bring with them privacy problems if their biological features are stored and used to track and profile people. For these reasons biometrics were only taken into consideration when used as one of the multiple authentication factors, and only when a trusted device (such as a smartphone) was used to collect them.

2.1 Smartcards/tokens

Smartcards are easy to use and have been around for a long time. They are currently widely used as citizen identification, as credit cards, to store information or to allow access to buildings or devices. But unfortunately it is common for each card to be designed for a single purpose or service, resulting in people having to carry many cards at the same time. Their use in authentication on computers or similar devices has also been hindered by the limited availability of readers, with passwords being the preferred alternate solution. The lack of a single standard also means that even if a reader was present, nothing guarantees that your cards would work on it. The necessity of special software drivers for the readers and the proliferation of mostly proprietary technology has also made difficult to use smartcards on web browsers as the main means of authentication.

In order to address these problems, a number of solutions have been proposed. One example of this is the Universal 2nd Factor (U2F)[10], which is an open authentication standard developed

by the FIDO alliance [8], a non-profit industry consortium created in 2012 and supported by large corporations such as Google and Microsoft. This protocol was designed to allow the use of security keys (smartcards with a male USB connector or NFC) as the second factor of authentication. This standard still requires the users to provide their credentials first, not solving the problems explained on the first chapter.

On [11] a solution has been proposed where a smartcard is used along with a hash function to encrypt a password and make it secure to be transmitted. While [12] proposes a method that allows smartcards to be used in order to temporarily establish trust in the keys stored in a computer's Trusted Platform Module, so it could be used to securely login while reducing the number of times the smartcard has to be read.

As far as authenticating to operating systems, certain Microsoft Windows versions allow smartcards to be used to login to user accounts[13], and support for USB security keys can also be achieved by installing extra software [14]. On the other hand, [15] and [16] shows examples of smartcard authentication on Linux based operating systems. In order for any solution to achieve widespread use when authenticating to operating systems, native support would likely have to be implemented by Microsoft, Apple and other large companies.

Biometric authentication on computers is also easily available today, with many laptops shipping with built-in fingerprint readers that can be used to unlock them. Using a single factor of authentication in this case is less problematic since an attacker would need access to both your computer and your fingerprint or security key, but it is not viable when authenticating through networks such as the Internet.

SIM cards have also been proposed as alternatives to store sensitive information. On [17] a method to store encryption keys on a SIM card is proposed, where a smartphone with NFC would be used in order to transfer the keys to a computer, which would store them in memory only for the duration of the authentication process, protecting them when they were not in use. This method has the clear disadvantage of allowing malware on either the computer or smartphone to be able to gain access to the user's private keys.

Other solutions involving both smartcards and smartphones have been proposed. [18] discusses a method that would allow a smartphone with NFC capabilities to be plugged into a computer so it could be used as a NFC smartcard reader. [19] describes a more complex solution that require both a smartcard and a mobile phone that can read QR codes. It has very bad usability since it not only requires a device to read the smartcard, but also prompts the user to read a QR-code with a smartphone, which then exhibits a generated number that needs to be typed back on the device he is authenticating to.

A commercially available solution, called "ekaay NFC"[20], uses smartphones with NFC as smartcard readers, getting the login challenge from the target device by reading a displayed QR-code. The smartphone then sends the challenge to the smartcard and receives the response through NFC, which is then forwarded through the Internet to the appropriate server. This solution seems secure and easy to use, but not only it requires both a NFC enabled phone and smartcard, it also does not fully disclose it's code or protocols. By not being fully transparent, the user ends up

having to trust that this solution has no backdoors or security flaws, which is not ideal. Ekaay also has other simpler versions that do not use smartcards and will be mentioned on section 2.2.

The popularization of embedded Secure Elements, which are similar to smartcard chips integrated with the hardware of smartphones or other devices, are also bound to make solutions that use a separate smartphone and smartcard much less appealing. The only disadvantage of merging these two technologies is that malware on the smartphone could now be used to access the cryptographic keys at any time, which wouldn't happen if you kept your smartcard separated from your phone. Though less secure[21], Host card emulation could also be used to give any device with an internet connection the same capabilities as secure elements.

2.2 Smartphone applications

With payment solutions such as Apple pay and Google wallet being introduced, and with the steady increase in mobile banking[22], the use of smartphone applications for security and authentication has become more common. In the last few years a large number of applications have appeared that promise to allow users to authenticate to a variety of different services, while increasing both security and usability. Most of these applications use challenge/response algorithms, using either a QR code reader or NFC to get the challenge, while usually sending the response back through the internet. In this section I will quickly present the main features of a few of these applications and protocols, showing their advantages and possible problems.

The solutions found [23, 24, 25, 26, 27, 28, 29] that are not commercially available and that lack enough documentation will not be listed. Also please note that the information bellow might not be entirely accurate since most of that information was either extracted directly from their creator's websites or was quickly inferred during the limited time spent testing each of them. A more serious and lengthy analysis of these solutions is out of the scope of this work, with the creation of the list below only having the goals of finding the best available protocols and applications and of allowing a better understanding of the usefulness of my own created solution, described in chapter 3 and 4.

Also note that solutions that use smartphones merely as a second factor of authentication, while still relying on passwords, will not be discussed. This is because, as previously explained in the fist chapter, typing passwords on an untrusted device is neither secure nor private. On the other hand, typing a password (or PIN number) for local authentication on a trusted device, such as a smartphone, was considered secure enough as long as this password is not the only factor of authentication.

Below is the list of solutions that involve smartphones, with each item having a brief explanation, followed by a summary list of their good and bad features and characteristics:

- **[1]TIQR** — Is an open source authentication solution for smart phones and web applications. As shown in Figure 2.1, it works by first using the app to scan a QR code (1), which contains a random challenge and other information. Then the user confirms the login (2)

and inputs his PIN number on the device (3), which finally accesses the TIQR server on the internet and logins through a challenge/response algorithm that utilizes symmetric cryptography.

- **Good:**
  - Uses challenge/response algorithms.
  - Is free for users and services.
  - Two factor authentication.
  - Open source SDK and application.
- **Bad:**
  - Since it uses shared keys with OCRA's OATH Challenge-Response Algorithm, the server database might have enough information to impersonate a user if it is breached.
  - The only implementation of TIQR that utilizes RSA cryptography still authenticates with the symmetric keys, using the private keys only to digitally sign information.[30]
  - Though it has been integrated in a number of identity federation solutions, it is not clear in how many sites it can be used.

Figure 2.1: TIQR login process[1]

- **[31]Clef** — Is a solution that uses a proprietary application to read a kind of animated barcode, which they call the "clef wave", in order to authenticate the user through a challenge/response algorithm using the internet. Figure 2.2 shows the animated barcode, containing bars of different sizes that move up and down. After reading the barcode the user is prompted to insert a PIN number on the app, which then uses public key cryptography to sign a challenge and authenticate the user.
  - **Good:**

    * Uses public key cryptography with challenge/response algorithms.
    * Server does not have enough information to impersonate a user.
    * Allows remote logout in case of theft of phone or computer.
    * Claims to be usable on over 140 thousand sites.
    * Is free for users.
    * Uses two factor authentication.
    * Has an open source WordPress plugin.
  - **Bad:**
    * There is no open source application.
    * Websites and services need to pay to implement it.

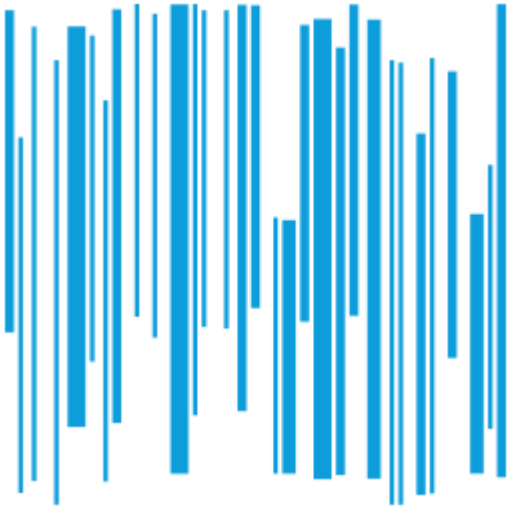

Figure 2.2: The "Clef Wave"

- **[32]eKaay** — Is a collection of proprietary protocols that use a smartphone to read a QR code in order to authenticate the user through a challenge/response algorithm. It allows login through many different variations. The "eKaay PIN" variation has the unique advantage of preventing malware on either the smartphone or the computer from capturing both the private keys and the PIN. This is done by displaying a numeric keyboard with scrambled numbers on one device, while the user has to type their PIN number on the other (on a blank keyboard). Another standout variation is "eKaay Sign", allowing the smartphone app to digitally sign any text, encoded in the QR code, which could be used to allow a number of things ranging from signed messages to secure bank transfers. "eKaay NFC" offers the greatest security, as mentioned on section 2.1, by storing the keys on a smartcard which is accessed through NFC. The final two variations, "eKaay PIN light" and "eKaay light", use any QR code reader application to read a code with a URL that opens a web page on the smartphone, where either a PIN (scrambled similarly as on "eKaay PIN") or a password has

to be entered, counting on the fact that it is less likely for the password to be stolen on the user's device. The "light" versions will not be taken into account on the summary below.

  - **Good:**
    * Uses public key cryptography with challenge/response algorithms.
    * Server does not have enough information to impersonate a user.
    * Is free for users.
    * The same application can be used to login with multiple versions that have different characteristics, making it more likely to fit most service's needs.
    * Uses two factor authentication.
    * Has an available SDK to integrate protocol in other applications.
  - **Bad:**
    * Closed application could hide backdoors or security flaws.
    * Accepted in a handful of sites.
    * Information about the application code and protocol is not easily available.
    * Websites and services need to pay to implement it.

- **[33]Tozny** — Proprietary application that uses public key cryptography and authenticates using challenge/response algorithms to allow a user to login to a website by reading a QR code. The application is also able to receive push notifications that allow it to confirm or deny certain actions. They also allow new applications to be developed and integrated with their API.

  - **Good:**
    * Uses public key cryptography with challenge/response algorithms.
    * Server does not have enough information to impersonate a user.
    * Is free for users.
    * Uses two factor authentication.
    * Allows API integration into other apps/solutions.
    * Browsers can store information that is used to avoid the step of reading the QR code, having only to input the PIN in the app to login.
  - **Bad:**
    * Accepted in a limited number of sites.
    * Information about the application code and protocol is not easily available.
    * Closed technology could hide backdoors or security flaws.
    * Websites and services need to pay to implement it

- **[34]SQRL** — Secure Quick Reliable Login is an entirely open and free protocol designed to allow authentication using public key cryptography. In it a single master key is combined with the service's URL domain and the user pin/password in a hash function in order to create a unique RSA key pair for each service, increasing security and privacy. The authentication is done by using the private key to digitally sign a URL, obtained by reading a QR code or by clicking on a link, which contains a random challenge and the site's domain, among other information. The signed information is sent to that URL with an HTTPS POST query. The protocol also allows for the backup of the master key as an encrypted QR code, which can be stored or imported to other devices, as long as the encryption password is known. They claim that by using "Scrypt" [35], which is a "sequential memory hard" function, the encrypted master key is resistant to brute force attacks, preventing attackers from obtaining the original key without having the password.

  - **Good:**
    - Uses public key cryptography with challenge/response algorithms.
    - Server does not have enough information to impersonate a user.
    - Is free for users and services.
    - Uses two factor authentication.
    - Each key pair is specific to a single service's domain, which increases security and prevents tracking users.
    - By allowing users to safely backup their master key as a QR code encrypted with a password, exporting your keys to other devices or doing backups is fast and easy.
    - Has mechanisms to securely revoke and replace a lost identity/key, making the stolen identity useless.
    - Has desktop applications for Linux and Windows, which require the user to click on the QR code (which links to the same URL encoded in the image), allowing you to login with the same keys as those stored on your mobile devices,
  - **Bad:**
    - Since the protocol is not yet considered finished, current implementations might have to be updated to fit the final version.
    - The increased complexity added by the various ways users can export, backup and revoke their keys makes implementing it somewhat harder than other proposed solutions, while also making it difficult to analyze and look for loopholes. Figure 2.3, where "IMK" stands for "Identity Master Key", shows how complicated the key management can be in the protocol.
    - The current available smartphone applications are neither open source nor finished.
    - It is not not yet supported by any service or website.

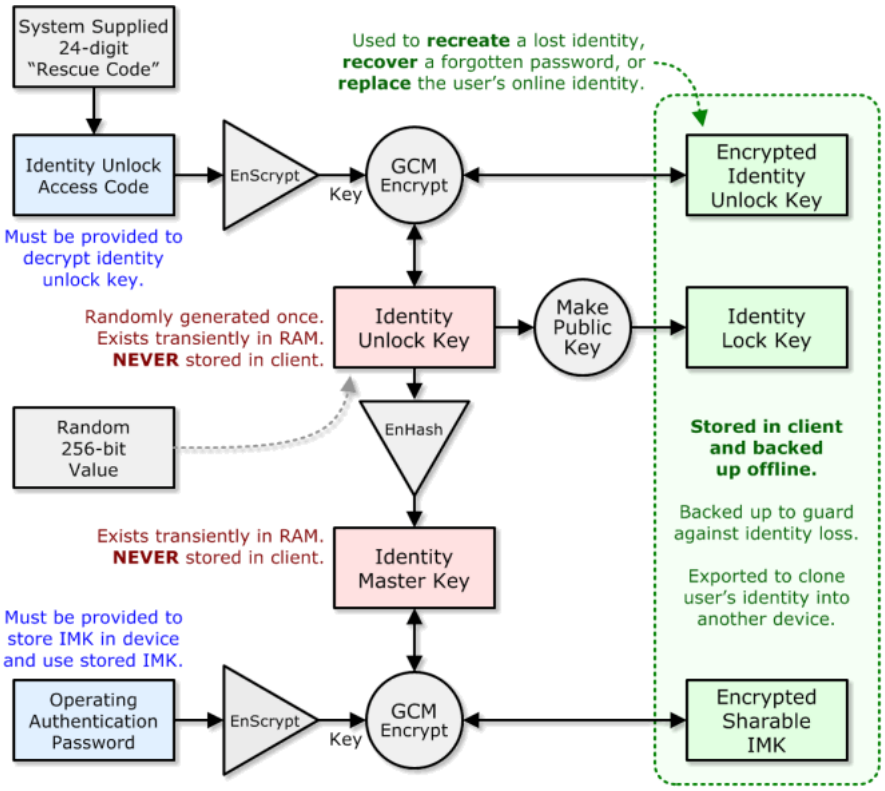

Figure 2.3: SQRL Client-Side Key Management[2]

- **[36]MYDIGIPASS** — According to their website, "MYDIGIPASS is VASCO's cloud-based, two-factor authentication service that enables application providers to deliver a single and secure login experience for their users, across multiple web and mobile applications". Their mobile application allows login by reading a QR code or by typing in a one time password, and also allows Single Sign On between their websites.

  - **Good:**
    - Is free for the users.
    - Available SDK to integrate the protocol into other applications.
  - **Bad:**
    - Uses several proprietary technologies that cannot be replicated or easily analyzed for security flaws.
    - Uses OAUTH 2.0 standard, which can contain a number of vulnerabilities if not well implemented[37].
    - Server probably has enough information to impersonate a user.
    - No open source application.
    - Implemented in a limited number of websites.
    - Websites and services need to pay to implement it.

- **[38]Google phone sign in** — Google has recently given users the capability of using their smartphones to login to their Google accounts on a computer. After typing in your username/e-mail on the website, a notification appears on your phone, and if it is confirmed the browser is automatically logged in. Users require either an Android smartphone with their Google account logged in, or an Iphone 5 or above with the Google app installed.
  - **Good:**
    - Is free for users.
    - Has the advantage over other two factor authentication methods offered by Google because your password is never in danger of being stolen.
    - No need to scan a QR code.
  - **Bad:**
    - It is still in testing stages when used without requiring a password to be provided first, and the details about what protocols or technologies are used are scarce. Users have to trust that this method is secure.
    - Can only be used to login to a Google account.
    - Since a notification appears on your phone whenever your Google e-mail is entered on their login page, anyone could send you a constant string of notifications, creating a sort of denial of service attack that also negatively impacts the experience of using your device.
    - Only available for use without passwords (not just a second factor) for the few users that were selected for testing.
    - You still have to type your e-mail on the browser, which could be used to track you.

Most of these solutions use QR codes as a one way channel to obtain information from the device they want to authenticate to because, unlike Bluetooth or NFC, cameras are available on the majority of smartphones on the market. They also rely on "out of band" authentication through the Internet, which is useful when authenticating to a terminal you do not trust. The majority of them also do not give a big emphasis to backing up a user's keys or identities, often relying on using the old password system as an alternate way to login in order to replace the lost keys for new ones.

But the most important conclusion taken from this list is that the majority of these applications are closed solutions that are not free to be implemented on a website, and that they are not interoperable. This, along with the fact that there are so many different solutions, could result in users having to adopt multiple applications with different protocols if they want to stop using passwords in most of their accounts.

On the other hand, the few free solutions encountered lack the marketing and customer support that would be required in order to convince most companies to abandon their existing password infrastructure. This has resulted in the even lower adoption rates of these open solutions, where

companies might often prefer to use paid implementations that are marketed as secure and that already have a certain userbase, instead of adopting a free but uncertain one.

In order to try to fix this fragmentation the FIDO alliance has recently released the Universal Authentication Framework (UAF) specification, which hopes to be a free and standardized protocol to enact passwordless authentication using strong public key cryptography. Since it is designed to work with many types of local authentication, such as biometrics or PIN codes, it allows users to choose how to locally authenticate to their devices that carry their private keys, making sure that any secrets (such as keys or biometric templates) never leave their devices. This standard could be used to greatly improve the interoperability between solutions, helping to popularize passwordless authentication. The fact that this standard is open and free could also help to popularize the solutions that make sure that the users have full control over their identities.

2.3 Comparisons

In this section a comparison between the proposed smartphone applications and protocols will be shown. The Google phone sign in will not be taken into account since it is not yet a solution that can be replicated on any service. Since this is a comparison between two factor authentication solutions, passwords will only enter the comparison when paired with a hypothetical smartphone application that allows for a second factor through the generation of a one time password. Each solution will be analyzed and scored in topics, with the results being displayed in two tables. The possible scores are bad (-), good (+) or very good (++). A quick explanation of each topic will be given first.

Usability and deployability topics, as displayed on table 2.1:

- **Id recovery** — The method directly offers a way to recover your identity if you lose your authentication keys or your smartphone.
- **Login speed** — How fast a user would take to login.
- **Memorization** — How much information a user has to remember in order to login.
- **Simplicity** — How easy it is for a user to learn how to use a solution, as well as how complicated the authentication process is.
- **Server secrets** — How much private information the server has about the user credentials. A bad score means the server has everything needed to impersonate the user. A good score means the server stores encrypted (or hashed) information that could be used in order to impersonate a user. A very good score means the server can identify a user with only a unique random Id and/or a public key, and that it doesn't store the required information to impersonate the users.
- **Nothing to carry** — How much hardware the user has to carry to login.

- **Free** — If the solution is proprietary and has a cost to implement, it has a bad (-) score, and if it is free to implement it has a very good (++) score. Passwords with a second factor might be more expensive to a service if it uses SMS messages, or if the OTP generator is not free.

- **Adoption** — How many services and websites that solution can be used to authenticate to. This information is not very accurate due to the difficulty of finding updated and reliable data on the subject. Good (+) scores in this case should not be interpreted as "good enough", simply indicating that they are not as good as passwords.

Table 2.1: Usability and deployability comparison

Topic	TIQR	eKaay	SQRL	Passwords + OTP	Tozny	Clef	MYDIGI-PASS	FIDO UAF
Id recovery	-	-	++	-	-	-	-	-
Login speed	+	+	+	+	+	+	+	+
Memorization	++	++	+	-	++	++	++	++
Simplicity	+	+	+	+	+	+	+	+
Server secrets	-	++	++	+	++	++	++	++
Nothing to carry	+	+	+	+	+	+	+	+
Free	++	-	++	+	-	-	-	++
Adoption	+	-	-	++	+	+	+	+

Topics regarding the solution's protection against attacks, as displayed on table 2.2:

- **Session theft** — When an attacker steals the one time password given to a browser after the authentication has been completed.

- **Computer Keyloggers** — When an attacker intercepts, either via malware or a compromised keyboard, the characters typed in the computer or device you are authenticating to. This can be a problem if the password intercepted can be used to access another account with the same credentials that is not protected by a second factor. This is the major problem for passwords, and is not totally solved by adding a second factor.

- **Smartphone Keyloggers** — When an attacker intercepts via malware the information input on the smartphone screen. Ekaay PIN uses a method to prevent the user's PIN from being stolen, while FIDO UAF, TIQR and Clef are confirmed to allow biometrics, which require no PIN to work. SQRL does not allow biometrics and require a password to unlock the private keys. The other solutions might not allow biometrics and could result in a stolen PIN, which still receives a good score since that is not enough to access the account.

- **"Shoulder surfing"** — When someone, malware or a camera is able to see what is being done on both the computer and smartphone screens in real time. In this case if an attacker can somehow steal the password and the one time password sent to the user, before the user uses it, this could be used to enter the user's account and steal it.

- **Brute-force** — When an attacker attempts to login multiple times by testing every possible combination. Passwords or one time passwords are usually a lot easier to guess than a response to a well done cryptographic challenge.
- **Social Engineering** — Any situation, such as phishing, where an attacker could convince a user to give him his credentials in order to login. There have been cases where an attacker stole a user's mobile phone number through social engineering in order to get the one time password, while phishing can easily steal a users regular password. SQRL has encryption keys that can be exported while protected by passwords, which could also be compromised. The other solutions prevent the keys from ever leaving the smartphones, preventing the identities from being stolen (justifying a very good score), but the users might still be tricked somehow into authenticating an attacker into their accounts, instead of themselves.
- **Server data leak** — When the secrets stored in the server are stolen and used to access user accounts.
- **Smartphone theft** — Since all solutions analyzed require the phone or the app to be locked by a pin or biometrics, they all receive good scores. Passwords have a higher score since they are often not stored in the device.
- **Key theft** — If an attacker manages to access the private or shared keys stored in the smartphone. The keys or seed used to generate the one time passwords could be stolen, but cannot be used alone to cause a breach if the password itself remains secure. SQRL claims to always store the master key encrypted with the user password, only storing the plain-text version momentarily in memory, making it harder but not impossible to steal. The other solutions could also reduce the risk by storing the keys in a secure element, but so could the SQRL keys.
- **User control and trust** — Whether the user has full control over his identity and can have full knowledge of the authentication process. Proprietary solutions that hide the authentication process from the website you are trying to login, forcing that site to ask their servers if a user should be granted access, receive a bad score. Solutions that are not fully documented (protocol, server and application code not available) or where the server has enough information to impersonate a user also receive a bad score. Passwords receive a bad score because a user has to trust the device he is typing them on. It is also worth noting that SQRL and UAF are protocols that do not yet have final and completely open implementations, so their score is based on a hypothetical and fully open application being developed in the future.

If we consider "User control and trust" as a necessity for authentication, only SQRL and FIDO UAF protocols would be good enough. But as already mentioned, neither of them have current applications that are free and easily available for users. This means that no solution has yet been developed that matches this criteria, justifying the creation during this thesis of a new application

that can fulfill it, while also being easy to integrate and to replicate. This application and protocol will be described in chapter 3 and 4.

Table 2.2: Security comparison

Topic	TIQR	eKaay	SQRL	Passwords + OTP	Tozny	Clef	MYDIGI-PASS	FIDO UAF
Session theft	-	-	-	-	-	-	-	-
Computer Keyloggers	++	++	++	-	++	++	++	++
Smartphone Keyloggers	++	++	+	++	+	++	+	++
Shoulder Surfing	+	+	+	-	+	+	+	+
Brute-Force	++	++	++	+	++	++	++	++
Social Engineering	++	++	-	-	++	++	++	++
Server data leak	-	++	++	+	++	++	++	++
Smartphone theft	+	+	+	++	+	+	+	+
Key theft	-	-	+	++	-	-	-	-
User control and trust	-	-	+	-	-	-	-	+

Chapter 3

My developed solution: Design

Considering the results presented on chapter 2, it was clear that there were solutions such as FIDO UAF or SQRL that could be used to achieve the main objective of giving users full control of their identities. The problem with those protocols is that they not only didn't have a proper implementation that is free, but that they also might be too complex for most users to understand, or for smaller websites to have the money to adapt and implement them. For this reason it was decided that it made sense to develop another solution. For a number of reasons, it was also decided early on that it would utilize a smartphone to read a QR code, and that it would use public key cryptography. Using smartcards or other tokens to store the encryption keys was ruled out due to the added complexity of communicating with them, and also due to the extra costs for users. This chapter will present a list of the requirements that were taken into account when designing my solution, followed by a summary of the solution and explanations about the design decisions that were taken during the development.

3.1 Base requirements

When designing the solution, the following base requirements were taken into consideration:

- Make sure the protocol is simple but secure, using the least number of messages as possible.
- Use RSA public key cryptography for authentication.
- Use symmetric keys to secure data during the transmission, reducing the used bandwidth and preventing interception or malicious alteration of the data in transit.
- Make sure the application and server code both work on as many devices as possible, reducing software and hardware requirements whenever possible.
- Use QR codes to obtain the challenge from the target device.
- Avoid protocols that rely on trusting third parties or certificate authorities (such as Transport Layer Security).

- Make sure the protocol and application are completely free by using only open source technologies.
- Comment all code created and make it available on the Internet, allowing it to be analyzed and implemented with the least amount of effort. Users and services should be able to check the code for backdoors and compile it for themselves.
- Confirm a second factor of authentication is used on the user's smartphone, by either requiring a PIN to open the application or forcing the use of a similar security factor on the device's OS.

3.2 Summary of solution

The developed solution utilizes an Android application (app) that is used to read a QR code that contains a random challenge, a session Id and an IP/port. The server then confirms the login attempt and it's name is displayed on the app screen, requesting a confirmation from the user to continue the authentication. If confirmed, the app uses it's unique RSA private key to sign the challenge obtained from the QR code. This signed challenge is sent along with the session Id and the user's public key to a Java server through the internet, using the IP and port from the QR code to create a simple Java socket. Any sensitive information in the protocol messages is sent encrypted by an AES shared key that is sent from the server to the user during the registration process. If the signed challenge, public key and session id are all validated by the server, a confirmation is received by the app, who informs that the user can now log in on the device. The session Id then becomes a sort of one time password that can temporarily give access to the user's account. The user then clicks on the QR code (if authenticating on a browser), which sends the session Id to the server in an HTTP POST form, just as it would usually send a username and a password, authenticating the user. This login process is shown on Figure 3.1.

Figure 3.1: Login process [3]

Registration of the app also happens by scanning a QR code that is only accessible either on the creation of the user account, or after accessing a secure page on the user's legacy account management page. This special QR code has the differences that the session Id is replaced with the username from that account, and that it has an extra field that contains an AES shared key. The

application reads that QR code and sends a message encrypted with that shared key to the Java server, resulting in the app's public key being associated with that account, allowing it to be used to authenticate that user. The random challenge in this case acts as a "password" that allows the application to associate it's public key with that user account. Since the shared key and public keys are exchanged during registration, this solution avoids using certificate authorities, only requiring the use of HTTPS to securely obtain the registration QR code if it is sent to a web browser. A password is also asked when registering, and it could either be a password specific to the service or the user, or it could be empty and not used at all.

The interaction between the Java server and the service or website that is using it to allow QR code authentication is done asynchronously through entries on a database. Just as a web server would usually check on a database if a username and password pair exist, it would now simply have to both check if a session id has been authenticated, and to translate that session id into a username so it knows what account to give access to. This means that adding QR code login to a web page would require minimal alteration on the login process, with the work consisting mostly in adding a couple database tables, installing the Java server with the database password, adding the code to translate a session id to the correspondent username and creating the pages that show the registration and login QR codes.

3.3 Design (architecture)

3.3.1 Application

The application was designed with 3 different screens. The first one on Figure 3.2 is the login screen, and is the first to open. The text "FeupQR" shown is the example name of the currently selected service, which was previously registered. Tapping on it opens a list with the registered services, which have to be properly selected in order to use the correct shared key, or the authentication will fail. In order to login the user taps on the "LOGIN" button, which opens the screen shown on Figure 3.3, used to read the QR code.

After reading a correct code and connecting to the server, the confirmation screen shown on Figure 3.4 appears. If the user taps "CANCEL" the app goes back to the login screen, but if the user taps "OK", the challenge is signed with the private key and is sent to the server. If the authentication is accepted, the screen on Figure 3.5 is displayed, containing the session id, the time of the login, the service name and the logout button, used to cancel a login before the QR code is clicked.

The registration screen is shown on Figure 3.6, and allows the user to type in a password. This password could either be specific to the user, to the service, or be left blank. This password could be, for example, the actual password of that account, preventing the registration QR code from being used if intercepted. By tapping on "START SCAN TO REGISTER" a screen similar to the one on Figure 3.3 appears, which the user has to utilize to read the registration QR code provided by the service.

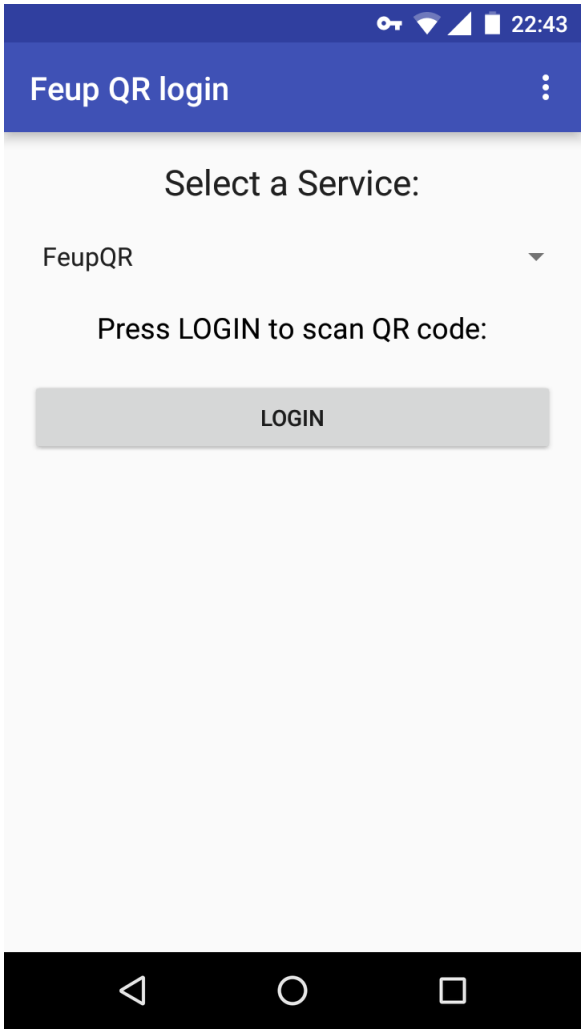

Figure 3.2: Login screen

Figure 3.3: QR code scanner screen

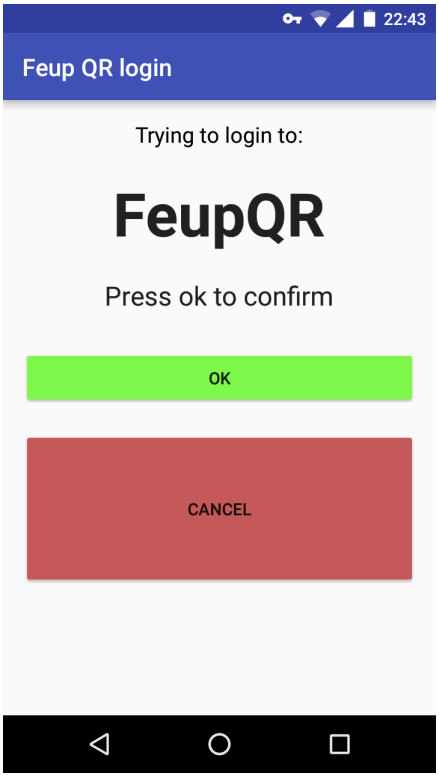

Figure 3.4: Confirmation screen

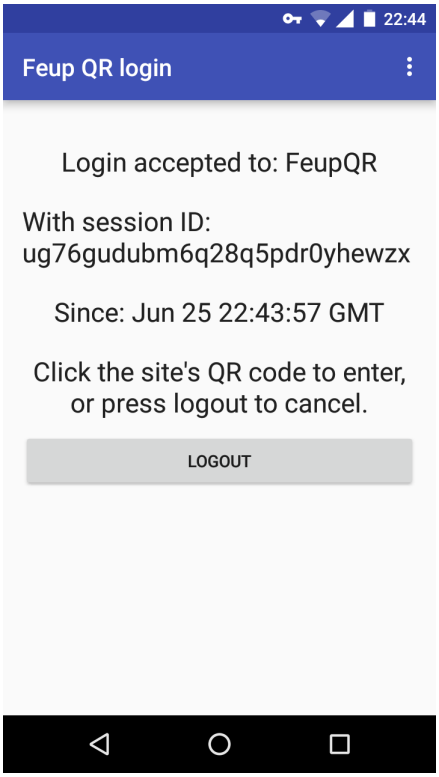

Figure 3.5: Confirmed authentication screen

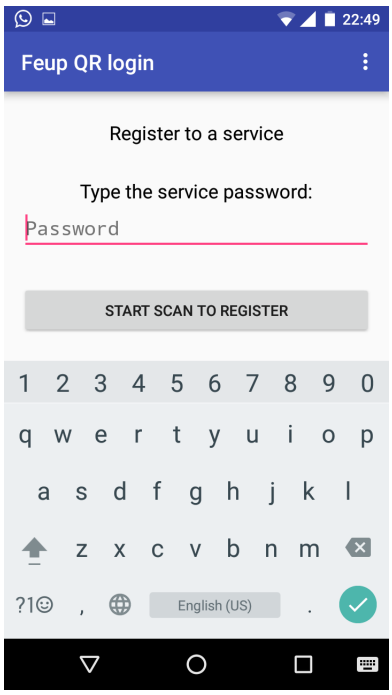

Figure 3.6: Registration screen

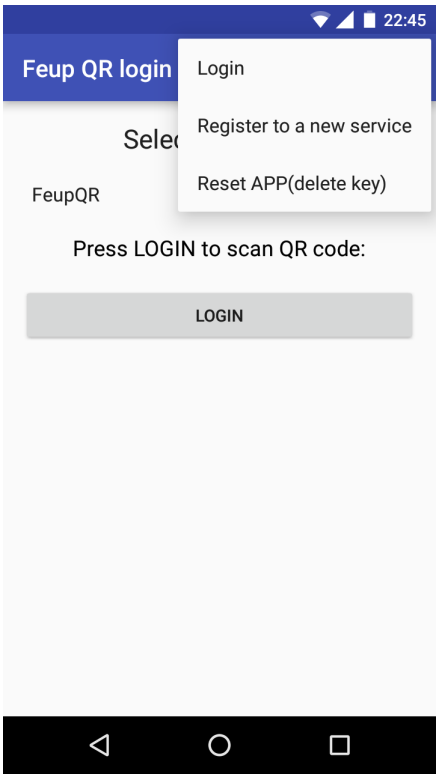

Figure 3.7: Menu opened on the login screen

The final screen shown on Figure 3.8 is used to reset the application, deleting all information and creating a new RSA key pair. The login, registration and reset screens are accessible through the application menu on the top right, shown on Figure 3.7. The confirmation screen only appears if a valid login message response is received, and the confirmed authentication screen only appears after a valid confirmation message response is received. These messages will be explained on subsection 4.1.

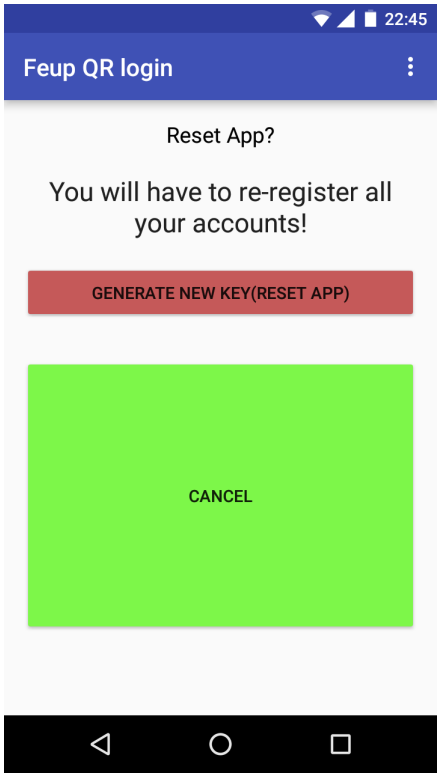

Figure 3.8: Reset screen

3.3.2 Used protocols and technologies

The solution uses a variety of existing protocols and technologies that are common in the industry. RSA public-key cryptography is used in order to safely encrypt some information before transmission, but was mostly utilized to allow for the application to authenticate without requiring the server to know the user secret. AES cryptography, with a shared secret known to the server, is used to secure the information that is transferred between server and application, and was chosen instead of RSA for securing the data in transfer while using less bandwidth and processing time.

The application was programmed for the Android operating system because it is a widely used system that has easily available tools to work with. Android Studio was used for programming and compiling the code. The application's RSA private key was stored using Android's keystore system in order to make it as hard as possible to extract, even with access to the unlocked device. The server was programmed in Java not only because it would allow it to work on a variety

of operating systems, but also in order to reuse as much code from the Android application as possible. The example implementation utilizes a PostgreSQL database, but the code could be quickly altered to use different databases since all SQL queries are relatively simple.

Since the Android keystore only works with some form of smartphone lock enabled, there was no need to add a separate PIN or other method to lock the app. The application tests if the smartphone lock is enabled, and will instruct the user to enable it before working. The use of the keystore also facilitates the future use of embedded secure elements to secure the keys, increasing security. The user's private key also cannot be easily extracted even by the user, helping to prevent phishing attacks and misuse.

The QR code scanner was provided by the ZXing ("zebra crossing") image processing library[39], which is open-source and available in both Java and other languages. The same library was used to generate the QR codes on the demo web page used to test the protocol. QR codes were chosen as the method to get information into the device because most smartphones have cameras that can be used to read them, unlike NFC technology, and also because it is both quick and easy to make sure you are only getting the information from the screen/device you want to authenticate to. The use of multiple devices that can authenticate to a single account is something a service would have to allow and prepare for, and would be done by having multiple RSA key pairs associated with the same account. If a private key is somehow lost or stolen, the lost identity could be recovered if an e-mail or phone number was previously added to the user account in order to use a method similar to current reset password solutions.

3.3.3 The QR code

The login QR code encodes a string in the form of a URL with the following composition:

feupqr://[IP]:[port]/auth?&terminal=[sid]&nonce=[n]

The parts surrounded by brackets are replaced on each QR code and contain the information required to login. The URL parts are explained below:

- **feupqr://** — Indicates the protocol, temporarily called FEUP QR.
- **[IP]** — Contains the server's IPv4 address.
- **[port]** — Indicates the port the server listens to.
- **[sid]** — Indicates the random alphanumeric identification of that QR code, or session Id, and is later used as a one time password to login.
- **[n]** — The alphanumeric challenge that has to be signed by the user's private key.

An example of a valid URL encoded in a QR code can be seen below, separated in two lines so it fits the page:

feupqr://192.168.0.105:4444/auth?&terminal=orhbpgqyhjvix50m4s9hpv8madzz
&nonce=emwf6d5yv5q40eh9in79w67ym6

This URL, encoded as a QR code, can be seen on Figure 3.9.

Figure 3.9: Login QR code example

The QR code used for registration is slightly different. A representation of the new URL is shown bellow, followed by an explanation of the different parts:

feupqr://[IP]:[port]/auth?&terminal=[register+username]&nonce=[n]&shared=[skey]

- **[register+username]** — The word "register" is concatenated with the username of the account you are registering your application to. This could be used by the application to exhibit the name of the user and ask if it is the correct one, but this is not done in the current application.
- **[skey]** — The shared secret that will be used to encrypt the information in the registration messages. It is important to use a secure channel to obtain this QR code, or this secret could be intercepted and used to decipher the registration messages.

An example of a registration URL is shown below, separated in two lines so it fits the page.

feupqr://192.168.0.105:4444/auth?&terminal=orhbpgqyhjvix50m4s9hpv8mad
&nonce=emwf6d5yv5q40eh9in79w67ym6&shared=tfmlDHCvgkPgp/j/t7ZIzA==

That URL can be seen encoded as a QR code in Figure 3.10.

Figure 3.10: Register QR code example

Chapter 4

My developed solution: Implementation

This chapter contains details about the developed protocol and the application and server code, as well as explaining the interaction between that server and the service (such as a web page server). In the end an analysis of the results are also presented, comparing this solution to the ones mentioned on chapter 2.

4.1 App and server interaction

The application communicates with the Java server with simple Java sockets, connecting to the IP and port encoded within the scanned QR code. This means that there is no problem for a server to change their IP or port, as long as that change is reflected on the QR codes displayed. It also prevents the client from requiring to access DNS servers, speeding up the process. The solution uses a small custom text based protocol, with each line being a single message. Information is sent in separated slots, with only 4 possible messages that can be sent to the server, whose response to each being either a confirmation message or a numbered error code.

The four possible protocol messages sent by the application are listed below. The underlined parts of the message are previously encrypted with the shared secret that was obtained during registration, which is stored in the app and used on the authentication messages.

- **Registration message** —
  register=nonce%%2%%pkey%%3%%SignedPass%%4%%randomN%%5%%pass

  Where "SignedPass" is the server or user specific password that was encrypted using the users private key ("pkey"). "nonce" is a random alphanumeric string and "randomN" is a random number to prevent an attacker from tricking a user application into encrypting a "nonce" that is actually a valid message, using their shared key.

- **Login message** — login=servername%%1%%pkey%%2%%sid%%3%%nonce

  Where "servername" is the name of the service, "pkey" is the user's public key, "sid" is a session identification alphanumeric string and "nonce" is a random alphanumeric challenge.

The user's public key is sent in plain text to allow the server to select the corresponding AES secret to decode the rest of the message.

- **Confirmation message** — conf=sessionId%%1%%nonceSigned

  Where "sessionId" is the same as "sid" on the "Login message", and "nonceSigned" is the "nonce" encrypted with the user's private key. Note that in order to create a message that would be considered a correct answer, in a way that the server would accept this session Id as a one time password that gives access to a user's account, the application not only has to know the correct "nonce" for that session Id, but also possess both the shared and private keys of the user in order to encrypt the nonce and the whole message.

- **Logout message** — out=conId%%1%%conIdSigned

  Where "conId" is a nine digit connection identification number obtained as a response from the login message, and "conIdSigned" is that number encrypted with the user's private key.

Each message has a single response from the server, with the difference that the registration message and login messages initiate a connection to the server, while the server responses to the registration message and logout messages close the connection. Below are the possible responses sent by the server:

- **Registration message response** —
  okR1=SName%%1%%ServerKey%%2%%sharedKey%%3%%randomN

  Where "SName" is the name of the service, "ServerKey" is the server's public key, and "sharedKey" is the AES shared secret the server has assigned to that user. In this message the underlined part is actually encrypted with the public key sent by the user in the registration message. That way only the owner of that public key would be able to know the secret. "randomN" is an added random string to prevent replay attacks, but it might be useless and can be omitted. "okR1" is replaced with "okR2" if that same public key was already registered on that service, and can be used to warn the user. A second registration is allowed and has the advantage of changing the shared secret between the server and the app.

  The following self explanatory error codes can be sent instead if something went wrong:

  error001=Wrong password

  error002=Key already registered

  error003=Wrong register message

  error004=Register error

  error005=Wrong signature

  error006=Wrong protocol message

  error007=Nonce not found

  error008=Not a register QR code

- **Login message response** — okL=conId%%1%%sid%%2%%servername%%3%%nonce

  Where "conId" is a nine digits number that identifies the connection, "sid" is the session Id received from the client, "servername" is the server's name and "nonce" is the random alphanumeric challenge received from the app.

  The following error codes can be sent instead if something went wrong:

  error000=Unknown connection key

  error006=Wrong protocol message

  error100=Key not registered

  error150=Invalid login

  error200=Terminal not found

  error201=Invalid nonce

  error202=invalid login message

  error203=Wrong server name

  error211=Register QR

- **Confirmation message response** — okC=conId

  Where "conId" is the nine digit identification of that connection.

  The following codes can be sent instead if something went wrong:

  error300=Unknown connection

  error400=Authentication failed

- **Logout message response** — okO=conId

  Where "conId" is the nine digit identification of that connection.

  The following codes can be sent instead if something went wrong:

  error500=Unknown session

  error600=Wrong signature

Each service could assign a different meaning to each error code, though the currently developed application assumes certain meanings in order to show an explanation of the code on the screen.

4.2 Server and service interaction

The Java server talks to the service by creating entries in a database table, which can then be read asynchronously by the service. Since it is common for a service to already have a database they access to check for their user's passwords, the changes necessary to allow for QR login with this

solution are minimal. This also means that this solution could be used to authenticate to any device that can access a database and display a QR code.

In the registration process the Java server adds an entry to the registration table on the database containing the user's public key, the user's username and the shared secret between the server and the user. During the authentication process, the service creates a session Id and a challenge, and stores them in an entry in the database that also contains the current time to prevent old challenges from being used. After the authentication process is completed and the application is authenticated, the Java server adds an entry to a different table, containing the session Id, the authenticated user's public key and the current time. For a small amount of time after this entry is created, anyone can use that session Id as a sort of one time password to enter that public key's user's account. This can easily be done, for example, by clicking on the scanned QR code, which would send the session Id in a HTTP POST form. The service then checks the table containing the correlation between the session Ids and public keys, obtaining the user's public key. That key is then searched in the registration table on the database, which contains the correlation between a user's username and his registered public key, obtaining what account it should display.

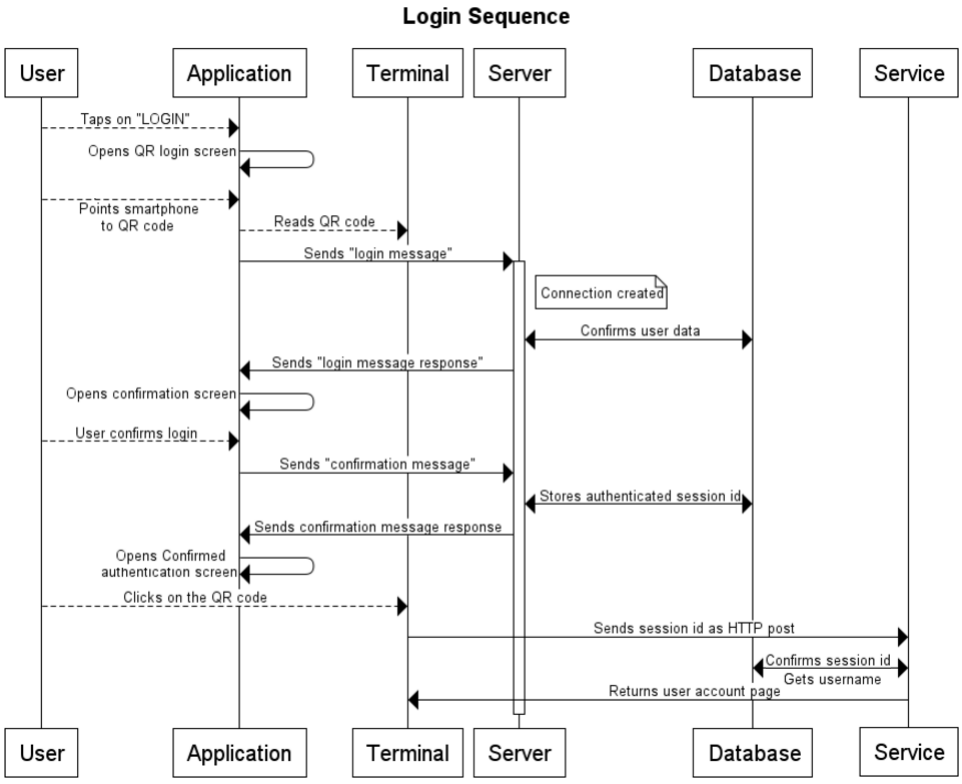

Figure 4.1: Sequence diagram of a login using the app

Figure 4.1 shows a sequence diagram that explains the login process, and makes it easier to see how the server and service communicate indirectly through the database. In this example the "Terminal" is a web browser the "User" is authenticating to, using his "Application" that was previously registered. The "Server" is the Java server previously mentioned, and the "Service"

is the web page's server. Filled lines with a single arrow are messages sent through the internet, dashed lines are actions taken, while double arrows mean one or more SQL queries and commands.

4.3 Android application code

The Android application was developed entirely in Java, and is comprised of nine Java classes. Each class will be briefly explained bellow:

- **ConfirmLoginAct.java** — Class used to create the confirmation screen, send the confirmation message and deal with the confirmation message response.
- **ConnectionManager.java** — Class responsible for creating, storing and closing the connection to the server.
- **EncryptionClass.java** — Class that contains the encryption methods, along with methods to create or delete the RSA keys from the Android's keystore.
- **IntentIntegrator.java** — Class taken from the ZXing project, without modifications, that contains the methods required to start the QR code scan process (opening the QR code scanner screen) and to obtain the scan results.
- **IntentResult.java** — Class also taken from the ZXing project, without modifications. An object of this class is used to store the results of a QR code scan.
- **LoggedAct.java** — Class used to create the confirmed authentication screen, send the logout message and to process the logout message response.
- **LoginAct.java** — Class used to create the login screen, initiate the login QR code scan, send the login message and process the received login message response. If no valid RSA key is found upon the creation of an object of this class, a new RSA key pair is created by calling a method from the "EncryptionClass" class. It is also responsible for checking if the device is protected by a PIN or similar security measure by calling the method "keyguardManager.isKeyguardSecure()", and blocking the app's functions if it isn't.
- **RegisterAct.java** — Class used to create the registration screen, initiate the registration QR code scan, send the registration message and process the received registration message response.
- **ResetAct.java** — Class used to create the reset screen and reset the application by erasing all information and deleting the keys from the Android keystore.

4.4 Java server code

The server Code is made entirely in Java. In order to facilitate development a simulated terminal and a configuration window were also created and open when starting, but can both be disabled

by running the ".jar" file with the "-nowindow" argument. After opening, the server tries to get from the database the configuration information, which includes the IP address, port, name and password. It then starts listening to the chosen port. When a client tries to connect on that port, it creates a Java socket and a new thread to handle the interaction, storing the connection information in an isolated object. The code is split into five classes, which will be briefly explained bellow.

- **Terminal.java** — Class used to create the simulated terminal screen, having the methods used to create and update the QR codes exhibited on that screen.

  The terminal screen simply contains a couple of QR codes and a text that changes to simulate a user being authenticated to it.

- **ServerThread.java** — Subclass of the Java "Thread" class, possessing the code that runs on the thread created for each user. It also contains one different method to handle the response for each of the four possible valid messages the server can receive.

- **ServerScreen.java** — Class used to create the configuration screen for the server. It contains the methods used to change the server configurations using that screen.

  The configuration screen can be used to change the server's name, password, IP address and port (both shown on the QR code). It also has buttons to both reset the tables containing the user's registration information, and to reset the table that stores the currently valid session Ids and random challenges (encoded in the QR codes).

- **ServerAndTerminal.java** — Class that initializes the server, checks the database for the configuration information and is responsible for accepting the client connections and creating new threads (ServerThread objects) to handle them. It contains methods used for encryption and to create RSA and AES keys.

- **Database.java** — Class that contains the methods used to interact with the database.

4.5 Demonstration web page

As already explained, the current implementation uses and Android application and a Java server, but a demo website was also created for tests. This website, which contains a simple PHP based exam and question manager with login passwords, was quickly modified to prove that the alterations necessary to support the proposed protocol are minimal, and that the method works. It now allows login using both regular username/password and by reading a QR code with the application.

In order to register an application and allow it to authenticate to an account, the user has first to login using his password and go to the "QR code app registration" page, that only opens if the user is logged in. As seen in Figure 4.2, the registration only contains a special QR code with the session Id replaced by the word "register" concatenated with the username, which in this example is "TEST USER", forming "registerTEST USER". Any application/smartphone that reads this

QR code on the registration screen will then be allowed to authenticate the user to login to that account.

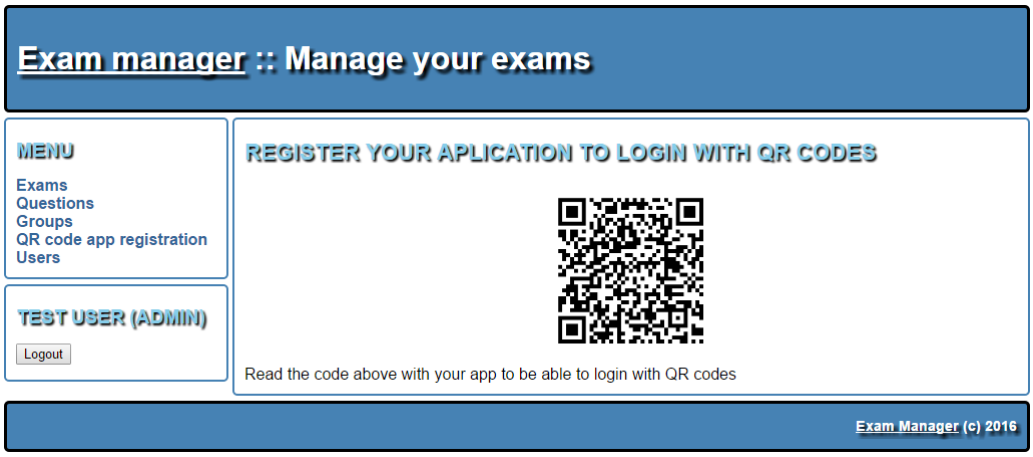

Figure 4.2: QR registration page

The login process involves clicking on the button "LoginQR", shown on Figure 4.3, which opens the page shown on Figure 4.4. This second page contains a QR code with a random session Id and challenge. If the user reads that QR code with his application, confirms the login on the smartphone and clicks on the QR code on the page, he will be logged into his account.

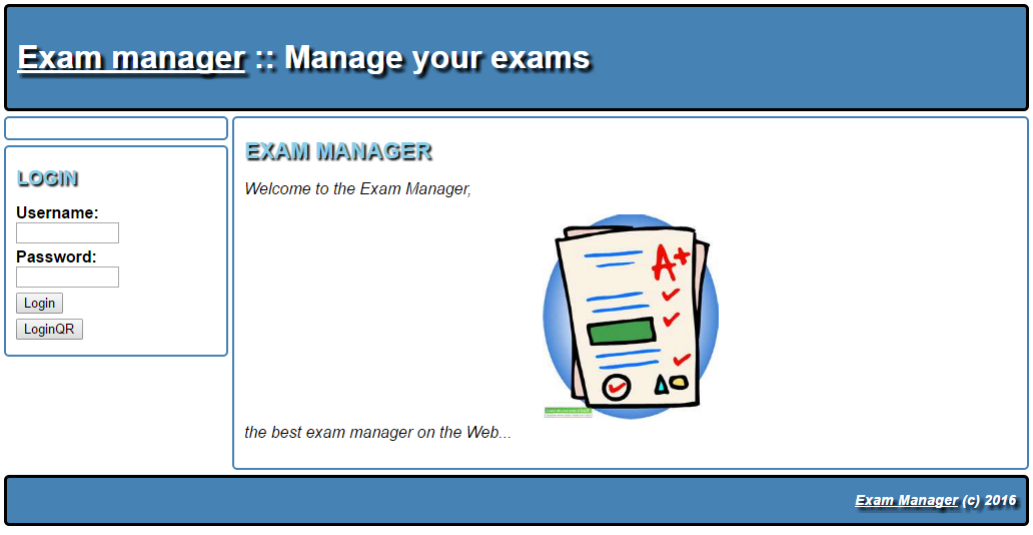

Figure 4.3: Login page

4.6 Solution analysis

A request to make a larger scale test by integrating this solution with one of the web pages from University of Porto was rejected by their system administrators, in part because there were no available servers that could be tested on, but also due to the small window of time between when

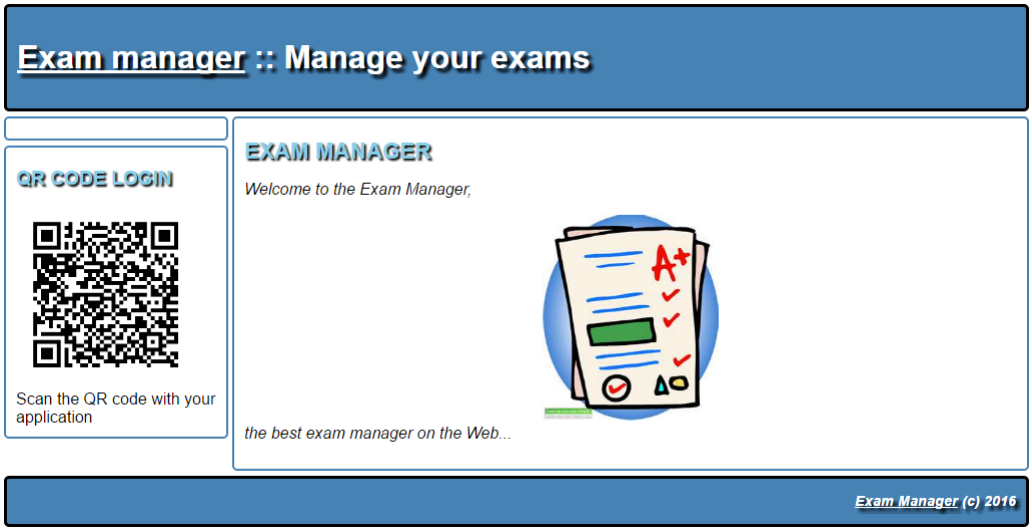

Figure 4.4: QR login page

the solution was ready and the end of the dissertation delivery deadline. As explained, a demo web page was still created and tested, but the solution was not analyzed in a real world scenario. In this section an analysis of the security and usability of the solution will be presented.

Just as most of the solutions investigated on chapter 2, the proposed protocol does not prevent session theft, because not only the authenticated session Id could be stolen, but it also does nothing to prevent the theft of the one time password stored in the user's browser. Also, even though the keys on this solution's app are stored in Android's keystore, there is nothing to prevent an attacker from using the keys if they were stolen. Though since the AES key is stored in the app's isolated storage space (that in theory can only be accessed by the app), while the RSA key pair is stored in the keystore (and in theory cannot be extracted), an attacker would have to gain access to both locations in order to be able to access a user's account.

This solution protects against computer keyloggers, with smartphone keyloggers or attackers using "shoulder surfing" techniques only being able to obtain the device's unlock PIN or password. Since the keys cannot be easily extracted from the keystore, an attacker using malware would only be able to access the user's account if it could extract both AES and RSA keys, which are stored in separate locations in the device, or by tricking the application itself into signing a challenge and a session Id known by the attacker.

Since the confirmation message requires both RSA and AES cryptography to respectively sign and then encrypt the random challenge, guessing the correct response is highly unlikely. It is more likely for an attacker to succeed by guessing session Ids, while sending HTTP forms to the server hoping to get lucky and use the right one in the small window of time it is accepted as a one time password to login to a user's account. But this is a lot harder than guessing regular passwords, because the session Ids are supposed to be completely random and have a lot more characters (28 characters are used in the demo web page).

This solution prevents phishing attacks from gaining permanent access to the user's account,

but it does not prevent an attacker from tricking a user into using the app to authenticate a QR code the attacker has access to, resulting in the attacker using the session Id to login to the user's account before him. A user should be careful not to read QR codes from websites that are not real, checking the website's certificates and encryption whenever authenticating on the Internet.

Though one of the base requirements used to design this solution specified that certificate authorities should be avoided, using certificates to verify the authenticity of a web page is a common thing and should be used if available. In case that the user is tricked anyway, he could still use the logout button to revoke the authentication entry on the database table, possibly disconnecting an attacker if that web page is designed to allow this. On the other hand, since a regular user shouldn't have the expertise to break into the Android's keystore, being tricked into sharing his keys with an attacker should be nearly impossible.

Smartphone theft is also not a problem as long as the attacker cannot unlock the device before the user can somehow use a side channel to revoke the keys in the device. This side channel could be the same as the one currently in use to change a user's lost password, which usually involves clicking on a link sent to your e-mail that contains a sort of one time password, allowing the user to choose a new password.

During a server data leak, the only things that could be stolen are the user's public and shared keys. But without having access to the private key (which never leaves the user's smartphone), there is no way for an attacker to use this information to impersonate the user. At most an attacker could discover what username corresponds to each public key, but the user accounts would remain safe.

The main security advantage offered by this solution is the fact that every aspect of it's protocol and application is well explained and freely available on the Internet. This means that a user could make sure that this solution has no backdoors, or that the server does not have information that could be used to gain access to his accounts.

When comparing the solution using the same usability and deployability topics found on table 2.1, the proposed solution would probably score similarly to the FIDO UAF protocol, but with a higher score on "Simplicity" and a lower score on "Adoption". This is because this solution has a protocol and application that try to be as simple and easy to understand as possible, and also because it is currently only implemented in the demo web page, which would give it a bad (-) score in "Adoption".

Login times using this solution were found to be as fast as 13 seconds when starting with a locked phone and a closed app, or 7 seconds if starting with the app open. Average times for a login without rushing, and starting from a locked device seem to be around 21 seconds. As a comparison, the fastest time achieved when using passwords and having to type in a one time password (received by SMS) was 17 seconds, and that was only achieved when copying and pasting both the username and password while reading the OTP from the locked phone screen. Average login time when having to type a one time password was around 35 seconds. Though these numbers were found with tests involving a single person and device, the large time different

(21 to 35 seconds) makes it likely that a larger study would also indicate that the proposed solution is faster.

With this information we can now add the proposed solution to the same tables found on chapter 2. The proposed protocol and application will be indicated by "QR" on both tables. Table 4.2 contains the security comparison, while table 4.1 contains the usability and deployability comparison.

Table 4.1: Usability and deployability comparison with proposed solution

Topic	TIQR	eKaay	SQRL	Pass-words + OTP	Tozny	Clef	MYDIGI-PASS	FIDO UAF	QR
Id recovery	-	-	++	-	-	-	-	-	-
Login speed	+	+	+	+	+	+	+	+	+
Memorization	++	++	+	-	++	++	++	++	++
Simplicity	+	+	+	+	+	+	+	+	++
Server secrets	-	++	++	+	++	++	++	++	++
Nothing to carry	+	+	+	+	+	+	+	+	+
Free	++	-	++	+	-	-	-	++	++
Adoption	+	-	-	++	+	+	+	+	-

Table 4.2: Security comparison with proposed solution

Topic	TIQR	eKaay	SQRL	Pass-words + OTP	Tozny	Clef	MYDIGI-PASS	FIDO UAF	QR
Session theft	-	-	-	-	-	-	-	-	-
Computer Keyloggers	++	++	++	-	++	++	++	++	++
Smartphone Keyloggers	++	++	+	++	+	++	+	++	++
Shoulder Surfing	+	+	+	-	+	+	+	+	+
Brute-Force	++	++	++	+	++	++	++	++	++
Social Engineering	++	++	-	-	++	++	++	++	++
Server data leak	-	++	++	+	++	++	++	++	++
Smartphone theft	+	+	+	++	+	+	+	+	+
Key theft	-	-	+	++	-	-	-	-	-
User control and trust	-	-	+	-	-	-	-	+	++

Chapter 5

Conclusion

In this thesis a solution was developed in the hopes of fixing the problem of users not having full control over their identities when using the current authentication protocols found on electronic devices. While it is clear that passwords are no longer secure enough for the digital age, an analysis of the currently available alternatives has also revealed that none of them have, so far, resulted in implementations that address all important issues. This chapter will present the overall conclusions on what was achieved with the work that was done, while also proposing a number of possible future improvements for the protocol and application that were developed.

5.1 Results

The initial assumption for this work was that there was no readily available way for users to authenticate themselves while still having control of the process. During the state of the art analysis it became clear that, while a number of solutions had already been proposed that could be used to remedy that problem, none of these solutions had become popular enough to be considered successful in replacing the regular authentication methods involving passwords.

For this reason a new solution was proposed, which involved the design and implementation of a communication protocol, a mobile application and a server capable of communicating with a database. With the hope of allowing this solution to be widely adopted, all parts of the developed solution were designed to use as much of the current password oriented infrastructure as possible, while also being as simple as possible to further reduce the costs of integration with current systems.

This design decision allowed the implementation of the solution on a demo web page in a very short amount of time, proving that the goal of small costs has been achieved. The necessary integration work will become even simpler now that the code has been made available in it's entirety with an open source license, allowing services to copy and reuse parts of that web page when integrating it on their own.

The use of QR codes as an alternative to NFC or other short range communication technologies was also deemed a success. The codes have proven to have more than enough capacity to store

the required information for the designed protocol, while also being reliable and easy to read. Since cameras are present in most smartphones available today, this has allowed the solution's application to be available for a larger number of users. This can also be confirmed by the fact that most commercially available authentication applications, among those that were encountered, also use some form of 2 dimensional bar codes as one of the methods to interact with other devices.

Even though a more extensive security analysis and protocol optimization might be necessary before it could be successfully implemented in a larger scale, it seems that the current state of the developed solution is close to being good enough to be used by smaller web pages and services that lack the money to implement one of the commercial solutions. Since this is also one of the few solutions that allow for complete modification and review of the code, which is relatively simple, it could also be modified to be used as a teaching tool or for other unexpected uses.

While the comparisons made between the proposed solution and the other available applications and protocols are not extensive enough to give an accurate answer, it seems that the current implementation is at least as good as most of them in the majority of analyzed topics. But still, FIDO's UAF protocol is the most likely one to become the accepted standard in the industry, mostly due to the considerable number of large corporations that support it.

5.2 Future work

While the protocol and developed code have currently reached a somewhat finalized proof of concept stage, a number of features and possible tweaks could be added in the future to increase it's security and appeal to the users. The protocol could be optimized to be smaller and more secure, for example, or the application could be ported as a desktop application that authenticates the user when opening the encoded URL with a mouse click. Since the code and protocol are available with open source licenses, these improvements could be done by anyone in the future. A list of some of the possible future features and improvements will be presented bellow:

- Add support on the application to allow it to store multiple keys and identities, along with an interface to manage the stored identities.
- Add support on the protocol to allow the application to read a different type of QR code that would be used to either confirm specific actions or sign data with the user's keys.
- Add the possibility of obtaining the URL, currently encoded in a QR code, by using NFC or Bluetooth technology.
- Add the explicit functionality (already possible with changes in server code) to allow the application to be used as a second factor of authentication, where the user's credentials would have to be typed and accepted before (or after) the QR code is scanned by the app. This would be safer and cheaper than sending one time passwords through SMS, while still being just as fast. This could also make the transition from passwords easier.

- Use hash functions to reduce the size of the protocol messages. The protocol could, for example, use a hash of the user's public key as the user's identification.
- Give a good name to the protocol, which could be something like "Simple trusted authentication protocol" (STAP), or "Secure Login Interface with Mobile phone" (SLIM).

References

[1] Tiqr. Tiqr website, 2016. URL: https://tiqr.org/ [last accessed 2016-06-27].

[2] Gibson Research Corporation. Sqrl client-side key management, 2016. URL: https://www.grc.com/sqrl/key-flow.htm [last accessed 2016-06-27].

[3] eKaay. Modified image from the ekaay website, 2016. URL: https://www.ekaay.com/Bilder/ekaayflowchart.jpg [last accessed 2016-06-27].

[4] Andy Adler. Vulnerabilities in biometric encryption systems. In *International Conference on Audio-and Video-Based Biometric Person Authentication*, pages 1100–1109. Springer, 2005.

[5] Anil K Jain and Ajay Kumar. Biometrics of next generation: An overview. *Second Generation Biometrics*, 12(1):2–3, 2010.

[6] Anil K Jain and Karthik Nandakumar. Biometric authentication: System security and user privacy. *IEEE Computer*, 45(11):87–92, 2012.

[7] Infineon. Security on nfc-enabled platforms building trust in the new mobile applications ecosystem, 2013. URL: http://www.infineon.com/dgdl/NFC_Whitepaper_09.2013_update_.pdf?fileId=5546d46149aa001a0149adc405ac003d [last accessed 2016-06-27].

[8] FIDO Alliance. Fido alliance overview on their website, 2016. URL: https://fidoalliance.org/about/overview/ [last accessed 2016-06-27].

[9] The Guardian. Hacker fakes german minister's fingerprints using photos of her hands, december 2014. URL: https://www.theguardian.com/technology/2014/dec/30/hacker-fakes-german-ministers-fingerprints-using-photos-of-her-hands [last accessed 2016-06-27].

[10] FIDO alliance. Fido alliance specifications overview, may 2015. URL: https://fidoalliance.org/specs/fido-u2f-v1.0-nfc-bt-amendment-20150514/fido-u2f-overview.html/ [last accessed 2016-06-27].

[11] Shyi-Tsong Wu and Bin-Chang Chieu. A user friendly remote authentication scheme with smart cards. *Computers and Security*, pages 547–550, 2003.

[12] Andreas Klenk, Holger Kinkelin, Christoph Eunicke, and Georg Carle. Preventing identity theft with electronic identity cards and the trusted platform module. In *Proceedings of the Second European Workshop on System Security*, EUROSEC '09, pages 44–51, New York, NY, USA, 2009. ACM. URL: http://doi.acm.org/10.1145/1519144.1519151, doi:10.1145/1519144.1519151.

[13] Microsoft. Security with smart cards, 2016. URL: https://technet.microsoft.com/en-us/library/cc962052.aspx [last accessed 2016-06-27].

[14] Yubico. Yubico windows login, 2016. URL: https://www.yubico.com/why-yubico/for-individuals/computer-login/windows-login/ [last accessed 2016-06-27].

[15] Debian. Smartcards on debian, 2016. URL: https://wiki.debian.org/Smartcards [last accessed 2016-06-27].

[16] CentOS. How smart card login works, 2016. URL: https://www.centos.org/docs/5/html/5.1/Deployment_Guide/sso-sc-login-concept.html [last accessed 2016-06-27].

[17] T. Mantoro and A. Milišić. Smart card authentication for internet applications using nfc enabled phone. In *Information and Communication Technology for the Muslim World (ICT4M), 2010 International Conference on*, pages D13–D18, Dec 2010. doi:10.1109/ICT4M.2010.5971895.

[18] Frank Morgner, Dominik Oepen, Wolf Müller, and Jens-Peter Redlich. *Mobile Smart Card Reader Using NFC-Enabled Smartphones*, pages 24–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. URL: http://dx.doi.org/10.1007/978-3-642-33392-7_3, doi:10.1007/978-3-642-33392-7_3.

[19] N Harini, TR Padmanabhan, et al. 2cauth: A new two factor authentication scheme using qr-code. *International Journal of Engineering and Technology*, 5(2):1087–1094, 2013.

[20] Bernd Borchert and Max Günther. Indirect nfc-login. In *ICITST*, pages 204–209, 2013.

[21] Pardis Pourghomi, Pierre E Abi-Char, and Gheorghita Ghinea. Towards a mobile payment market: A comparative analysis of host card emulation and secure element. *International Journal of Computer Science and Information Security*, 13(12):156, 2015.

[22] Board of Governors of The Federal Reserve System. Consumers and mobile financial services report, 2015. URL: http://www.federalreserve.gov/econresdata/consumers-and-mobile-financial-services-report-201503.pdf [last accessed 2016-06-27].

[23] Rawad Kilani and Kenneth Jensen. Mobile authentication with nfc enabled smartphones. *Technical Report Electronics and Computer Engineering*, 2(14), 2015.

[24] Young-Gon Kim and Moon-Seog Jun. A design of user authentication system using qr code identifying method. In *Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on*, pages 31–35. IEEE, 2011.

[25] Ben Dodson Debangsu Sengupta Dan Boneh and Monica S Lam. Snap2pass: Consumer-friendly challenge-response authentication with a phone.

[26] Ra Dmitrienko, Ahmad-Reza Sadeghi, Eep Tamrakar, and Christian Wachsmann. Smarttokens: Delegable access control with nfc-enabled smartphones (full version). 2012.

[27] T Venkat Narayana Rao and K Vedavathi. Authentication using mobile phone as a security token. *IJCSET*, 1(9):569–574, 2011.

[28] Ben Dodson, Debangsu Sengupta, Dan Boneh, and Monica S Lam. Secure, consumer-friendly web authentication and payments with a phone. In *International Conference on Mobile Computing, Applications, and Services*, pages 17–38. Springer, 2010.

[29] Peng Kunyu, Zheng Jiande, and Yang Jing. An identity authentication system based on mobile phone token. In *2009 IEEE International Conference on Network Infrastructure and Digital Content*, pages 570–575. IEEE, 2009.

[30] RCDevs SA. Tiqr login and signing server, 2016. URL: http://www.rcdevs.com/products/tiqr/n [last accessed 2016-06-27].

[31] Clef. Clef website, 2016. URL: https://getclef.com/ [last accessed 2016-06-27].

[32] ekaay. ekaay website, 2016. URL: http://www.ekaay.com/ [last accessed 2016-06-27].

[33] Galois company. Tozny website, 2016. URL: http://tozny.com/ [last accessed 2016-06-27].

[34] Gibson Research Corporation. Sqrl website, 2016. URL: https://www.grc.com/sqrl/sqrl.htm [last accessed 2016-06-27].

[35] Colin Percival. Stronger key derivation via sequential memory-hard functions. *Self-published*, pages 1–16, 2009.

[36] VASCO Data Security International. Mydigipass website, 2016. URL: https://www.mydigipass.com/ [last accessed 2016-06-27].

[37] Cnet. Serious security flaw in oauth, openid discovered, 2016. URL: http://www.cnet.com/au/news/serious-security-flaw-in-oauth-and-openid-discovered/ [last accessed 2016-06-27].

[38] Google. Google support website, 2016. URL: https://support.google.com/accounts/answer/6361026?hl=en [last accessed 2016-06-27].

[39] Official ZXing ("Zebra Crossing") project home. Fido alliance overview on their website, 2016. URL: https://github.com/zxing/zxing [last accessed 2016-06-27].